# Build System

## Background

A critical component of Agile Software Engineering is Continuous Integration loosely understood to be integration of individual efforts into the overall system frequently. At the center of such a notion is a build of the entire application whenever an individual engineer releases his/her work. Most teams can ill afford manual builds. There are quite a few build systems such as Jenkins (https://www.jenkins.io), circleci (https://circleci.com) that support such automation.

In this projectlet we will build the core of a build system. The key features then are:

- Retrieve the source code from a revision control system such as git
- Create the links necessary to trace the build to the source code
- Build the application
- Publish the build artifacts
- Archive the log of all the activity

## Requirement Specs

Let us specify the requirements for a build tool called jobs.

| Id | Need |
|---|---|
| 1 | Support git based repositories |
| 2 | Support distinct branches within git repositories |
| 3 | The tool be able to enumerate the branches at the time of initialization or later. |
| 4 | Each build shall be assigned a unique identification starting with 1 |
| 5 | The tool shall support a retention criteria - in terms of the number of builds. |
| 6 | The tool shall generate traceability information that can be included in the builds suitable for languages: Ada, C, C++, Python, and Go. The traceability information can also be generated as ini formatted text files. |
| 7 | Traceability information required:<br>Branch name<br>Commit id - brief and full<br>Build date and time<br>Build System ie host name where built |
| 8 | The tool shall support the following stages of builds: initialization, build, publish, archive.<br>Initialization - retrieve other repositories as necessary<br>Build - build the application<br>Publish - the build artifacts to be published<br>Archive - Capture the build history as an archive |
| 9 | The build stages can be driven by a script checked out from the repository |
| 7 | |

## Design Inputs

| Req ID | Command Line Example | |
|---|---|---|
| 1 | jobs init [options] git@bitbucket.org:ToprLLC/passwords.git | At the designated workspace, create the top level structure for this job. |
| 2 | —jobspace=<dirname> default: $JOBSPACE | Specify a folder as the jobspace. The repository name is used as the folder for this job's workspace; For the above repository the name will be passwords. |
| 3 | —init=scriptname<br>Optional | Name of a script file to be found in the repository that is used for the init stage of the builds |
| 4 | —build=scriptname<br>Optional | Name of a script file to be found in the repository that is used for the build stage of the builds. |
| 5 | —publish=scriptname<br>Optional | As above, a script for publishing the archives |
| 6 | —archive=scriptname<br>Optional | As above, a script for archiving build logs |
| 7 | —trace=(C,C++,Ada,Python,Go,Text) default: Ada | Trace information file to be generated in the specified language(s) |
| 8 | —retention=number default: 3 | |
| 9 | jobs enumerate jobname | For the specified example, the job name is passwords.<br>This commands enumerates the branches and creates a branch workspace. |
| 10 | jobs build jobname [branchname] | For the specified branch perform a build as the next build id. |
| 11 | —pull | Git pull before building |
| 12 | —script=scriptname | Execute the script from the repository. The default is to execute the job specifications in the order: init, build, publish and archive |
| 13 | —all | This switch requests a build of all the branches |
| 14 | jobs purge <job name> | Purge the build artifacts in conformance with the retention specifications |

| Req ID | Command Line Example | |
|---|---|---|
| 14 | jobs show | Complete log of all the builds |

## Learning Objectives

- ✓ Manipulate, navigate directory structures in a platform independent way.
- ✓ Creating, storing configuration files in a standard format - json
- ✓ Executing external commands and capture their output